

# CPXe: Web Services for Internet Imaging



**The Common Picture eXchange environment leverages the Web services paradigm to serve the electronic photographic services market, combining open standards for exchanging digital images, orders, and other information with an online directory of service providers.**

*Timothy  
Thompson*  
*Rick Weil*  
*Mark D.  
Wood*  
Eastman Kodak  
Company

**D**igital cameras promise instant gratification—you can take a picture and immediately print it or send it to others. In practice, however, consumers typically find manipulating, printing, and sharing digital photos frustrating, expensive, and time-consuming. Digital camera and software vendors offer various solutions to streamline the workflow, but these proprietary approaches often limit choice and flexibility, preventing users from fully realizing the benefits of digital photography and restraining overall industry growth.

Standardized film sizes, processing types, and photofinishing options have long enabled the traditional film-based industry to grow and thrive, with many vendors offering a rich variety of services. To provide similar support for the digital photographic services market, Kodak has joined with other industry companies—Agfa-Gevaert, Beaufort Solutions (formerly Gretag/Telepix Imaging), Fuji Photo Film, Hewlett-Packard, Konica, Noritsu, Olympus America, Pixology, Shutterfly, and SilverWire—to create the Common Picture eXchange environment ([www.i3a.org/i\\_cpxe.html](http://www.i3a.org/i_cpxe.html)).

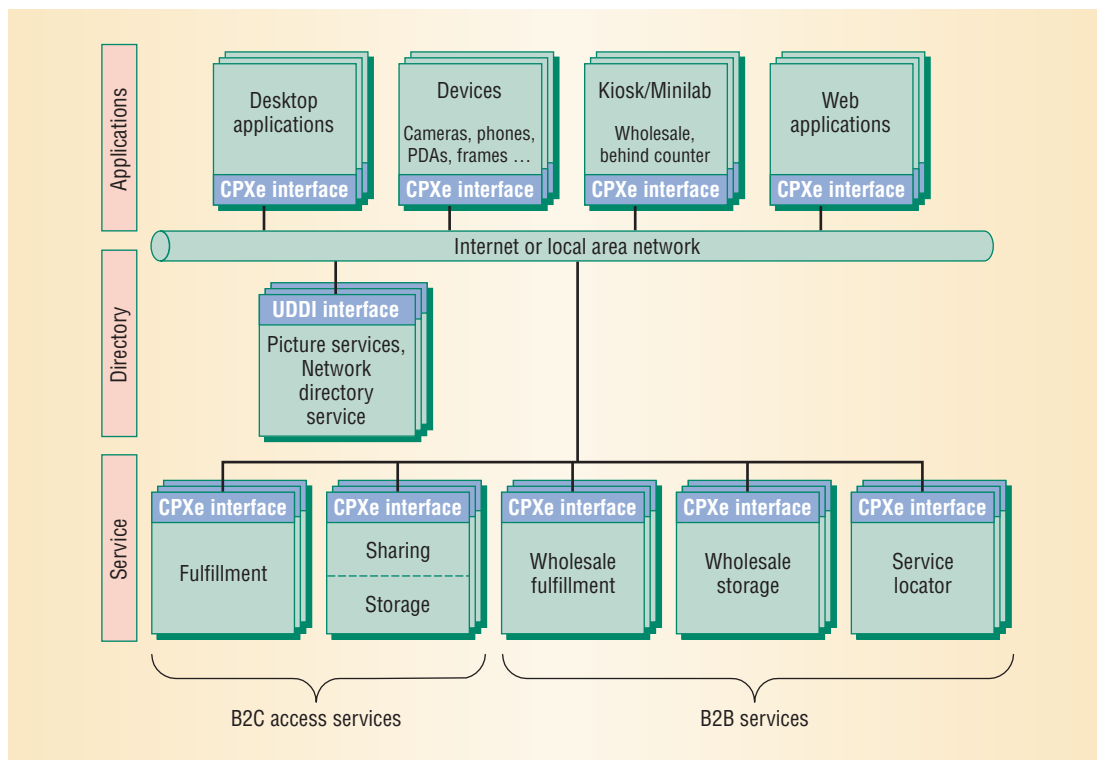
CPXe is a highly interoperable service delivery framework that leverages the Web services paradigm to give providers access to an expanded market and offer consumers a broad range of digital imaging services. Multiple providers can register their services in a central directory and precisely characterize their offerings using an extensible catalog and order model as well as a comprehensive

dictionary of terms. A service locator mechanism lets consumers easily select vendors offering the products and features they desire.

CPXe's initial focus is to give users the printing options they are accustomed to, with the additional ease and flexibility enabled by digital technology. Customers can submit orders using a browser or desktop application, a kiosk, or even a networked digital camera and have their pictures printed at any participating retail or online fulfillment service, with retail-based services offering both in-store pickup and postal delivery. CPXe also gives retailers the flexibility to directly provide a portal or end-user application or to partner with camera vendors or global portal providers.

## SYSTEM ARCHITECTURE

As Figure 1 shows, the CPXe architecture consists of three tiers: the services themselves, a directory service, and the applications that discover and interact with these two types of services. It anticipates a wide range of business-to-consumer (B2C) access services, wholesale business-to-business (B2B) services, and utility services for supporting functionality such as transaction logging. To date, the CPXe Initiative Group has only defined access services for online fulfillment,<sup>1</sup> with additional services such as storage and sharing planned for the future. Any conforming application—from a desktop application running on a consumer's home computer to a kiosk application installed in a retail environment—can invoke these services.



**Figure 1. Common Picture eXchange environment component architecture. Any conforming application can invoke the business-to-consumer access services, wholesale business-to-business services, and directory service.**

## UDDI

CPXe relies on the universal description, discovery, and integration specification ([www.uddi.org](http://www.uddi.org)) for directory functionality.<sup>2,3</sup> UDDI is a Web service that lets businesses discover one another and describe how they interact. It provides simple object access protocol interfaces for publishing entries and querying the UDDI registry, and it uses document literal encoding to pass XML-formatted data in SOAP messages.

UDDI relies on *tModels* to represent metadata. A *tModel* is defined by a name, a description, and an overview; how it is used is up to the definer. In CPXe, providers use *tModels* to classify their services and specify their supported interfaces. UDDI provides a relatively flexible mechanism for searching the directory to find services that categorize themselves using a particular *tModel*. Because UDDI is not designed to provide the fine level of detail about service offerings that consumers want, each provider also operates a catalog service that describes its own services and products.

## Service locators

The CPXe system also implements the *service locators* concept.<sup>4</sup> Functioning much as a travel agent or sales broker, a service locator consults the UDDI directory to determine available services and queries those services for catalog information. Applications seeking specific kinds of services and products can interact directly with a selected service locator service to identify an appropriate service provider.

Currently, providers deploy their own service locators to direct requestors to their services; how-

ever, the architecture also allows for independent service locators who may charge for their service and are not tied to a particular vendor. Although applications can interact directly with UDDI, most will interact with a specific service locator service. Portal and end-user application providers typically will implement the service locator service that their application uses. Vendors can enter into business agreements with specific service locator providers to be listed by that provider.

## DESCRIBING SERVICES

CPXe uses the Web Services Description Language<sup>5</sup> to describe the interfaces to all its services. Essentially an interface definition language expressed as an XML grammar, WSDL supports multiple type systems and different transport protocols including HTTP and SOAP, the mechanism for packaging requests and responses when interacting with remote services. SOAP 1.1<sup>6</sup> supports a variety of interaction models and data encodings.

The flexibility of WSDL and SOAP has resulted in interoperability problems between different vendors' Web service implementations. In response, the Web Services Interoperability Organization (WS-I) has developed Basic Profile 1.0<sup>7</sup> to define a preferred use model and specify what features Web service vendors must implement. Although Basic Profile 1.0 was not completed before development of CPXe, the system's designers were guided by early decisions made by the WS-I Basic Profile Working Group. CPXe adheres to most Basic Profile 1.0 requirements, including use of the document literal message format and SOAP binding in WSDL.

**The CPXe framework provides a standardized way to communicate detailed information about products and services.**

A fundamental CPXe function is to facilitate the exchange of product catalogs and consumer orders, which involves more than simply delivering messages to end points. In both types of operations, all parties to a transaction must have a common understanding of the data types, message formats, and service operations. At the same time, participants must be able to differentiate themselves by detailing their offerings and accommodating future extensions. WSDL service definitions reference the CPXe data model, passing data and catalog objects as parameters. Both data objects are defined using XML Schema.<sup>8</sup>

### Catalogs

CPXe provides a common catalog framework to facilitate development of B2C and B2B relationships. This framework gives a provider a standardized way to communicate detailed information about its products and services to a service requestor. The provider can customize such information based upon its locale or its relationship with the requestor. The framework supports internationalizable descriptions, vendor extensions, complex product descriptions, proxy and proprietary exchanges, and catalog subsection requests; in the future, the service definition may be extended to support XPath-based queries.

The CPXe Initiative Group is responsible for creating and maintaining the catalog schema as well as the CPXe dictionary of categories, properties, and choices. Each provider can use the framework and dictionary to create its own catalog to classify and describe its products and services. Service providers must implement the `getCatalog` interface to return catalog information to requestors.

The CPXe catalog contains seven basic grouping elements:

- *servicePropertyList*: a list of service properties, including general business information;
- *storeList*: information about each retail store, including street address and hours of operation;
- *productList*: a description of each product;
- *priceList*: pricing information;
- *categoryList*: a list of categories that classify products;
- *propertyList*: a set of properties that further describe categories and products; and
- *choiceList*: a set of choices for defining properties.

Business models and methods differ greatly among industry participants: Some operate online only, while others have retail outlet stores; some do business in a single language, while others support multiple languages. The service property section provides specific details of a service provider's business to requesting applications. The last three catalog sections form a dictionary of terms that vendors use to describe their product offerings in a standardized way.

CPXe defines a set of basic digital image product categories that include, for example, photographs, greeting cards, T-shirts, and jigsaw puzzles. When participating companies build their catalog, they classify each product according to one of these categories and use the property and choice elements to describe each offering's details.

If its product does not fit well into the existing CPXe classification system, a company can create its own dictionary with category, property, and choice definitions. If other companies find the definitions useful, then CPXe may incorporate them into the CPXe dictionary. As with any such system, CPXe uses categories and other terms that provide sufficient specificity for finding products of interest without risking misclassification.

Each product has a unique identifier, name, and description. A set of `infoUrls` lets a requesting application retrieve pictures of the product in different configurations. If the application creates the product from other items in the catalog, the definition may specify the component parts—for example, the definition could describe a framed print as a kit made up of a picture frame and print. Finally, a product definition includes a list of service properties and properties particular to that specific product.

The “Categories, Properties, and Choices in CPXe” sidebar illustrates how service providers might use the CPXe framework and dictionary to create a catalog description of a 4-inch × 6-inch print.

### Orders

A service requestor creates an order and submits it to the provider for processing. The CPXe data model defines `OrderType` as a complex structure that captures all information needed for the order. Like the catalog, an order organizes information into several sections. These sections include a list of image assets, products and quantities, pricing and packaging information, consumer and vendor information, and current order properties and status.

The order is a shared document that both the requestor and provider use. A requestor can complete the requestor's portion and submit it to the

## Categories, Properties, and Choices in CPXe

To understand how service providers use CPXe categories, properties, and choices to create a dictionary, consider the concept of *scaling*. A service provider may need to scale digital images in size to fit a given product, such as a 4-inch × 6-inch print. If the aspect ratios of the image and printed area do not match, the image can be scaled to fit in various ways, depending on whether or not the image is cropped.

CPXe provides a well-defined meaning for scaling:

```
<property id="Scaling">
  <name>Scaling</name>
  <definition>Scaling specification for an
    image</definition>
  <infoUrl>http://www.i3a.org/cpxe/100/en/
    dictionary.html#Scaling</infoUrl>
  <choiceList>
    <choiceRef xlink:href="#FitWithin">
      Fit Within </choiceRef>
    <choiceRef xlink:href="#ZoomToFill">
      Zoom To Fill</choiceRef>
    ...
  </choiceList>
</property>
```

CPXe also provides standardized meanings for terms such as *FitWithin* and *ZoomToFill*. For example, CPXe defines *FitWithin* as:

```
<choice id="FitWithin">
  <name>Fit Within</name>
  <definition>The entire image will be
    printed with no cropping.</definition>
  <infoUrl>http://www.i3a.org/cpxe/100/
    en/dictionary.html#FitWithin</infoUrl>
</choice>
```

The framework defines other available choices in a similar manner. Vendors offering the product category *Photographs* can use one of these terms to characterize their scaling behavior.

The following definition of a 4-inch × 6-inch print from Kodak's catalog illustrates how a product definition in the CPXe catalog framework builds off the same base type upon which properties and choices are built:

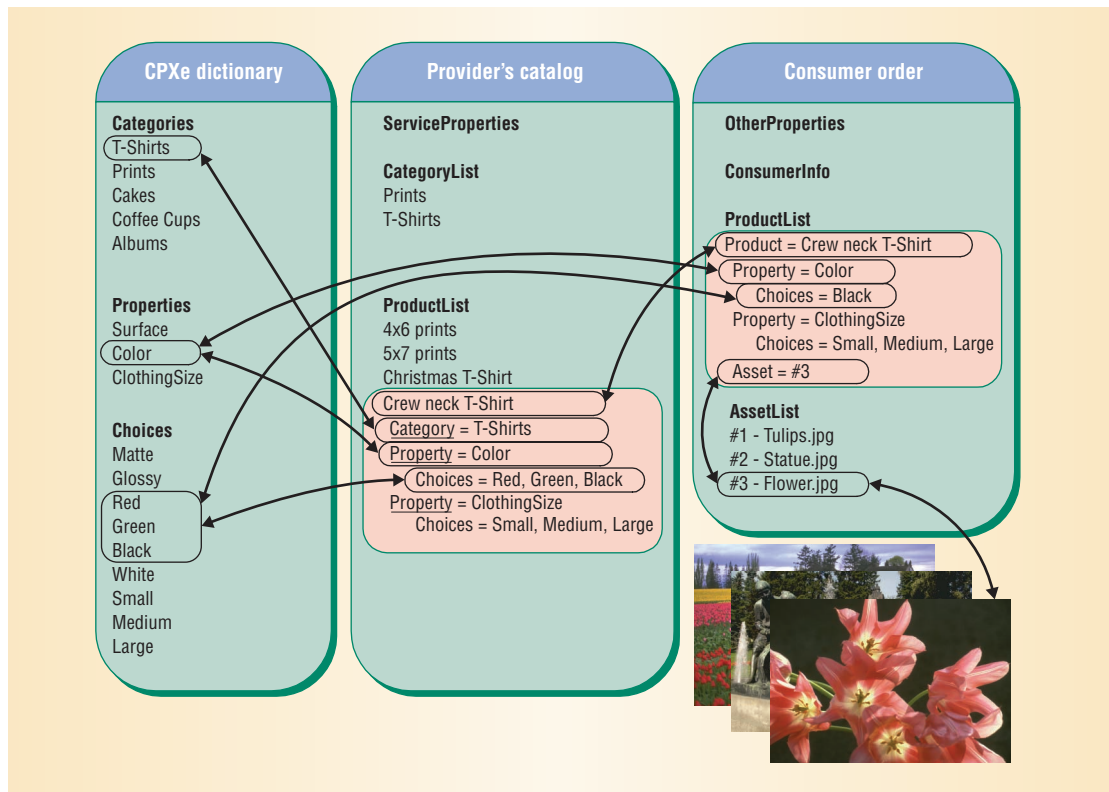
```
<product id="Prints4x6">
  <name>4x6 Prints</name>
  <definition>4x6 inch Kodak print with
    your choice of matte or glossy finish
  </definition>
  <infoUrl>http://cpxe.kodak.com/catalog/
    en/prints4x6.html</infoUrl>
  <categoryRef xlink:href="http://www.i3a.
```

```
org/cpxe/100/en/dictionary.
  xml#CatPhotographs">
    Photographs</categoryRef>
  <servicePropertyList>
    <expirationDate xlink:href="http://
      www.i3a.org/cpxe/100/en/dictionary.
        xml#ExpirationDate">
      <choiceValue>
        <value>2003-10-31T09:30:47-05:00
          </value>
        </choiceValue>
      </expirationDate>
    </servicePropertyList>
  <productPropertyList>
    <propertyChoiceRef link:href="http://
      www.i3a.org/cpxe/100/en/dictionary.
        xml#PhotoLength">
      <choiceValue>
        <value>10.16</value>
      </choiceValue>
    </propertyChoiceRef>
    <propertyChoiceRef xlink:href="http://
      www.i3a.org/cpxe/100/en/dictionary.
        xml#PhotoWidth">
      <choiceValue>
        <value>15.24</value>
      </choiceValue>
    </propertyChoiceRef>
    <propertyChoiceRef xlink:href="http://
      www.i3a.org/cpxe/100/en/dictionary.
        xml#Surface">
      <choiceRef xlink:href="http://
        www.i3a.org/cpxe/100/en/dictionary.
          xml#Glossy">
        <name>Glossy</name>
        <infoUrl type="IconImage">
          http://cpxe.kodak.com/catalog/
            en/prints4x6_glossy_icon.jpg
          </infoUrl>
        </choiceRef>
        <choiceRef xlink:href="http://
          www.i3a.org/cpxe/100/en/dictionary.
            xml#Matte">
          <name>Matte</name>
          <infoUrl type="IconImage">
            http://cpxe.kodak.com/catalog/
              en/prints4x6_matte_icon.jpg
            </infoUrl>
          </choiceRef>
        </propertyChoiceRef>
      </productPropertyList>
    </product>
```

provider or, alternatively, a requestor application can partially fill in the document and then hand it off to an optional *helper application* to complete.

This Web-based application provides the user interface for collecting consumer information such as a credit card number and address. After an online

**Figure 2. Relationship between a catalog and an order. The order, in this case a photo T-shirt, references the product description from the catalog, which in turn references the product definition from the dictionary.**



service has accepted an order and the local application has transferred the image assets to it for processing, the provider uses the order document to capture other information such as shipping charges and package tracking.

As Figure 2 shows, each element in an order's product list is linked to the service catalog's product definition. The product element also can contain a list of properties as defined by the catalog and manufacturing options. For example, if the catalog defines a print as being scalable in different ways, the product element can specify the scaling property and indicate the kind of scaling to apply. The product element also contains a list of references to the image assets needed to make the product. An asset's data model contains essential information that lets one or more products use it to track the status of its transfer from the local application to the online service.

## FINDING SERVICES

Service providers can use a UDDI directory server to publish their service offerings so that potential clients can search for them. The Picture Services Network, a subsidiary of the International Imaging Industry Association, administers a CPXe directory service (<https://production.pictureservices.org/directory/web>). To publish a service listing with the PSN directory service, PSN members create an XML document that conforms to the appropriate UDDI schema and then invoke the appropriate UDDI publish interface; members also can publish service listings using a browser-based interface.

Clients use the publicly available UDDI inquiry methods to search for services.

CPXe defines its services through the combination of the data model, expressed through XML Schema, and the service methods, expressed through WSDL. However, this formal interface specification does not capture all the information that a service provider needs to communicate to potential requestors. For example, consider a consumer who wants to know where she can get poster-sized glossy prints in Miami and each such service location's hours of operation. Answering such a query requires detailed data about each provider's products that would be difficult to provide and maintain through the tModel abstraction.

The consumer may further want to limit her search to places within five miles of a hotel. Such queries require a more sophisticated query mechanism than UDDI provides. In CPXe, end-user applications typically do not search UDDI directly through its query API. Instead, they communicate with service locators to potentially execute a more complicated search request. A service locator obtains basic information about available services from the PSN directory service and then queries providers directly to obtain more detailed data about their offerings.

## tModels

CPXe defines tModels to enable effective searching by service locators, which essentially function as intelligent agents that mine the UDDI database and the appropriate provider catalogs.

**Table 1. CPXe taxonomies.**

Name	Purpose	Key names	Key values	
ServiceClassification	Defines type of service	ServiceType	Locator FulfillmentAccess Application ExtendedLocator ExtendedFulfillmentAccess	
Language	Defines language support for services	ServiceLanguages	Language codes from RFC-3066 or ISO 639 in the form of 'll-cc' or 'll'	
		ApplicationLanguages	Language codes from RFC-3066 or ISO 639	
		DefaultLanguage	A language code from RFC-3066 or ISO 639	
FAServiceProperty	Defines various service properties of a fulfillment access service	ExcludeOrderCountries	Two-character country codes from ISO 3166, identifying countries in which a business or service is not willing to do business	
		OrderCountries	Two-character country codes from ISO 3166, identifying countries in which a business or service is willing to do business	
		ExcludeDeliveryCountries	Two-character country codes from ISO 3166, implying that a service is willing to deliver to any country <i>not</i> on the list	
		DeliveryCountries	Two-character country codes from ISO 3166, implying a service is willing to deliver <i>only</i> to countries on the list	
		Currencies	Three-character currency codes from ISO 4217	
		ProductCategories	CatPrints	CatMousePads
			CatPrintSheets	CatTshirts
			CatDrinkingCups	CatSweatshirts
			CatJigsawPuzzles	CatCakes
			CatDrinkingCups	CatCookies
DeliveryTimes	Standard	NextDay		
	OneHour	Express		
	SameDay			
DeliveryDestinations	PickupInStore DomesticDelivery InternationalDelivery			

UDDI describes two basic types of tModels.<sup>9</sup> A *taxonomy* tModel defines a mechanism for classifying various kinds of entities—typically a set of key names and allowable values for that key. A *technical fingerprint* tModel defines a particular interaction model, perhaps by using WSDL. CPXe defines a set of taxonomy tModels for classifying CPXe services and a set of technical fingerprint tModels for describing particular service types.<sup>10</sup> PSN directory service members use these tModels to classify their services and to specify supported interactions.

CPXe service locators use the taxonomy tModel to determine the type of service being offered and the properties of that service. The technical fingerprint tModel governs the interaction with the service, typically by identifying an interface such as a SOAP-based API or an HTTP interface.

**Taxonomies.** A vendor uses a keyedReference structure to place information about its business and services in a UDDI category bag. The keyed reference has attributes that refer to a key name, key value, and tModelKey. The tModelKey references the tModel that describes the semantics of the key name and value pairing.

A categoryBag includes references to CPXe taxonomies. Service locators use this information to answer requests such as, “Give me the fulfillment access services that make poster-size prints with one-hour service and offer in-store pickup.”

To enable searching, CPXe defines three taxonomy tModels:

- *classification*—to identify the type of service offered;
- *language*—to identify what languages a service can support; and
- *service properties*—to indicate where a service can do business, available delivery and pickup options, and what types of products the service offers.

Table 1 lists the key names and values associated with each of the taxonomies. FAServiceProperty corresponds to the fulfillment access service and currently is the only service property that CPXe defines. The framework may define additional properties for other types of services in the future.

Each CPXe tModel specifies a series of key names and their allowable values. The CPXe tModels are *checked taxonomies*—before the UDDI server accepts publication of an entry, it validates the entry’s use of the taxonomy. The PSN provides a validation service for each tModel. The UDDI server invokes the CPXe-defined validation service when a reference to a CPXe taxonomy occurs. The validator verifies the key names and determines that the corresponding key value is allowable as defined by the referenced tModel. The validator also ensures that each service has properly categorized itself by service type. If there is an error in the use of the taxonomy, the UDDI server denies the publication request.

Each taxonomy tModel has an associated uuid that is used when referencing the tModel. The fol-

**Using the technical fingerprint tModels ensures that each provider adheres to a standard profile.**

Following example uses a CPXe service's category bag to illustrate how the publisher references a tModel and uses it to classify a CPXe service:

```
<categoryBag>
  <!-- Classification tModel
  reference -->
  <keyedReference keyName="
  ServiceType" keyValue="
  FulfillmentAccess"
    tModelKey="uuid:AC35F67B-8DF1-
    492C-9BEB-0598F1CC69B9"/>
  <!-- Language tModel reference -->
  <keyedReference keyName="Default
  Language" keyValue="en"
    tModelKey="uuid:109D244D-
    D13E-4E49-8E6D-17FADB5363F1"
    />
  <!-- FAS Service Property tModel
  references -->
  <keyedReference keyName="Order
  Countries" keyValue="CA US"
    tModelKey="uuid:6dc64e8b-
    daa9-4c63-9ba3-fc0414094c6e"
    />
  <keyedReference keyName="Product
  Categories" keyValue="CatPrints
  CatMousePads"
    tModelKey="uuid:6dc64e8b-
    daa9-4c63-9ba3-fc0414094c6e"
    />
</categoryBag>
```

The first keyedReference refers to the Classification tModel, identified by the tModelKey attribute. This reference says that the service is type FulfillmentAccess. The second keyedReference uses the Language tModel to specify that the service's default language is English. The last two keyedReferences each refer to the FASServiceProperty tModel. The keyName OrderCountries explicitly lists all the countries with which this service can do business.

Entering proper classification information in categoryBag is essential for service locators to find the service. The UDDI server validates each of the entries using the PSN validator service. If, for example, a value entered in the keyValue attribute of OrderCountries is not a two-character country code from ISO 3166, the validator would generate an error, and the UDDI server would deny the request for publication. An invalid keyName would produce the same result.

**Technical fingerprints.** Each of the services that

CPXe defines adheres to a standard interface expressed in WSDL and described by a specific CPXe technical fingerprint tModel. Using the technical fingerprint tModels ensures that each provider adheres to a standard profile. A requestor can connect to different end points yet still use the same WSDL definition to interact with the service. To give fulfillment access service providers and application developers additional flexibility, CPXe defines several types of FAS technical fingerprints.

FASDirectClient is for FAS services that solely use a SOAP interface, defined by a WSDL document. The overviewURL in the tModel definition specifies the URL for the WSDL file, which an application can use to make SOAP requests to the accessPoint. Consider the following example:

```
<bindingTemplate bindingKey="
5BF604CB-FF6F-11D5-A2F2-
00C0F041184F"
  serviceKey="E9CA2607-F5C2-47CA-
  B9D0-EBE4D1D83E87">
  <accessPoint>http://FAS_soap
  </accessPoint>
  <tModelInstanceDetails>
    <tModelInstanceInfo tModelKey="
    uuid:812f5848-ee54-42c8-9c9a-
    d7f0eae6abde"/>
  </tModelInstanceDetails>
</bindingTemplate>
```

Any binding template that refers to the tModel for FASDirectClient supports the CPXe WSDL for that profile. An application can interact with any service provider end point that uses that profile, which gives requestors the ability to freely choose from among different conforming service providers.

FASHelperApp is for service providers who use a hybrid approach for online fulfillment, combining a SOAP-based API with a browser-based helper application. The helper application relieves requestors of some of the burden of communicating through a SOAP interface. Service providers are not required to support both FASDirectClient and FASHelperApp; the existence of both models gives application developers increased implementation flexibility at the expense of potentially decreased interoperability. Referencing FASHelperApp within a bindingTemplate is the same as referencing FASDirectClient, except the tModelKey attribute refers to the FASHelperApp uuid rather than the FASDirectClient uuid.

A third tModel is for services that support only a Web browser consumer interface and do not

expose fulfillment services through SOAP. However, to be included in the CPXe directory, these services must supply a SOAP interface for obtaining the product catalog. For service locators to function, the catalog must always be accessible via the CPXe SOAP interface.

## Service locators

CPXe service locators mask the complexity of interacting with the UDDI server by encapsulating the iterative queries and complex XML documents needed to do a UDDI search.

The service locators understand the semantics encapsulated by the CPXe tModels and use the inquiry API to query the UDDI server. They also can use information that other taxonomies provide, such as geographic data and information from other sources besides UDDI. Service locators use aggregated catalog information to build up a more complete description of the available services and products than would be available directly from UDDI itself—for example, price, product availability, and shipping options.

Service locators answer questions such as, “Where can I find a fulfillment access service that makes mouse pads and offers in-store pickup?” The UDDI directory contains sufficient information to answer some queries. The business service’s category bag uses keyed references to the appropriate tModel to describe this information. For example, a service’s directory listing might include the following:

```
<keyedReference keyName="Product
  Categories" keyValue="CatPrints
  CatDrinkingCups CatJigsawPuzzles
  CatMousePads CatT-shirts"
  tModelKey="uuid:6dc64e8b-
  daa9-4c63-9ba3-fc0414094c6e" />
```

However, because the UDDI query facility requires exact matches on key values, a service locator cannot use it to determine which listings include CatMousePads in their provided product categories. To determine which fulfillment access services produce mouse pads with in-store pickup, a service locator would first query the UDDI server to obtain a list of candidate access services, as the following call to the UDDI `find_service` method illustrates:

```
<find_service>
  <categoryBag>
    <keyedReference keyName="Service
      Type" keyValue=
```

```
"FulfillmentAccess"tModelKey=
  "uuid:AC35F67B-8DF1-492C-
  9BEB-0598F1CC69B9"/>
  </categoryBag>
</find_service>
```

This query will return a list of all fulfillment access services. The service locator then examines each service’s description to determine if its `ProductCategories` `keyedReference` contains the desired product, `CatMousePads`, and if its `DeliveryDestinations` `keyedReference` contains `PickUpInStore`.

By implementing a service locator that understands the corresponding tModels’ semantics, a provider can offer sophisticated search capabilities targeted at specific problems. For example, a service locator may be able to return a list of all stores that produce photo mouse pads and are within 10 miles of some address. UDDI does not, by itself, support such queries, but it does provide an infrastructure for building intelligent query agents. The key is understanding how tModels work and how to deploy them to solve specific problems.

## INTERACTING WITH SERVICES

Client applications interact with a service using a well-defined workflow. The interaction model uses sessions to maintain state across the multiple calls needed to place an order. To place an order with a fulfillment access service, a requestor minimally would use the following methods:

- *getSession*—to specify certain properties such as locale;
- *createOrder*—to create an `Order` object typically specifying customer information, product choices, payment information, and a list of image assets;
- *validateOrder*—to verify correctness of the order information and request computation of tax, shipping, and handling charges; and
- *authorizeOrder*—to submit the order for fulfillment and to authorize payment.

The appropriate technical fingerprint tModel describes the interfaces for these methods. Additional calls might be necessary to determine related data such as the service properties. Providers should supply secure socket connections to protect the confidentiality of data transmissions.

The order data structure does not include the binary image asset data. Instead, to upload an image file for remote fulfillment, the requesting applica-

**Implementing a service locator offers sophisticated search capabilities targeted at specific problems.**

tion creates an order and then retrieves properties of the order filled in by the service to determine the appropriate URL; next, the requesting application uses HTTP POST to upload the image to that URL. Current SOAP implementations do not provide an interoperable mechanism for efficiently including binary data in a SOAP message.

**T**he Web services model provides a sound basis for building and deploying interoperable services, allowing services to be described, registered, and used across the Internet. Current Web service protocol implementations do not always interoperate in practice; the recently released WS-I Basic Profile 1.0 profile will significantly improve interoperability by mandating use of XML Schema as the type system and by more tightly specifying allowable behavior.

Although UDDI provides an excellent foundation for intelligent search agents that understand tModel semantics, it is not intended for storing highly specific or dynamic information. The CPXe architecture combines UDDI with catalog services and service locators to provide a full-featured e-commerce directory for imaging services. By using the Web services model, CPXe provides the imaging industry with a flexible and extensible platform for offering consumers a wide range of product choices through a variety of interoperable service offerings. Future CPXe efforts will build on the current framework to support new service types and enhanced e-commerce capabilities. ■

---

### Acknowledgments

Andy Bailey, Howard Bussey, Lou Chauvin, Jack DeMarti, Tim Witcher, and the anonymous reviewers provided valuable feedback during the writing of this article.

---

### References

1. International Imaging Industry Association, "CPXe Fulfillment Access Service API Specification," I3A, 2003; [www.i3a.org/CPXe\\_downloads\\_1.html](http://www.i3a.org/CPXe_downloads_1.html).
2. D. Ehnebuske, B. McKee, and D. Rogers, eds., "UDDI Version 2.04 API Specification," UDDI Committee Specification, 19 July 2002; <http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm>.
3. D. Ehnebuske, D. Rogers, and C. van Riegen, eds., "UDDI Version 2.03 Data Structure Reference," UDDI Committee Specification, 19 July 2002; [http://uddi.org/pubs/DataStructure-V2.03-Published-](http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm)

[20020719.htm](http://www.i3a.org/CPXe_downloads_1.html).

4. International Imaging Industry Association, "CPXe Service Locator Service API Specification," I3A, 2003; [www.i3a.org/CPXe\\_downloads\\_1.html](http://www.i3a.org/CPXe_downloads_1.html).
5. E. Christensen et al., "Web Services Description Language (WSDL) 1.1," W3C Note, 15 Mar. 2001; [www.w3.org/TR/wsdl](http://www.w3.org/TR/wsdl).
6. D. Box et al., "Simple Object Access Protocol (SOAP) 1.1," W3C Note, 8 May 2000; [www.w3.org/TR/2000/NOTE-SOAP-20000508/](http://www.w3.org/TR/2000/NOTE-SOAP-20000508/).
7. K. Ballinger et al., eds., "Basic Profile Version 1.0," WS-I Working Group Draft, 24 Jan. 2003; [www.ws-i.org/Profiles/Basic/2003-01/BasicProfile-1.0-WGD.html](http://www.ws-i.org/Profiles/Basic/2003-01/BasicProfile-1.0-WGD.html).
8. H.S. Thompson et al., eds., "XML Schema Part 1: Structures," W3C Recommendation, 2 May 2001; [www.w3.org/TR/xmlschema-1/](http://www.w3.org/TR/xmlschema-1/).
9. B. McKee and D. Ehnebuske, "Providing a Taxonomy for Use in UDDI Version2," UDDI Working Draft Technical Note Document, 17 July 2001; [www.uddi.org/pubs/TN-taxonomy-provider-V1.00-Final-20010717.pdf](http://www.uddi.org/pubs/TN-taxonomy-provider-V1.00-Final-20010717.pdf).
10. International Imaging Industry Association, "Defined tModels and Profiles Specification," I3A, 2003; [www.i3a.org/CPXe\\_downloads\\_1.html](http://www.i3a.org/CPXe_downloads_1.html).

*Timothy Thompson is a software engineer at Kodak. His research focuses on XML technologies. Thompson received a BS in computer science from the State University of New York at Utica. Contact him at [timothy.thompson@kodak.com](mailto:timothy.thompson@kodak.com).*

*Rick Weil is an e-business architect at Kodak, working in the area of Web and networking technologies. He received a BS in electrical engineering from the University of Wisconsin. Contact him at [richard.weil@kodak.com](mailto:richard.weil@kodak.com).*

*Mark D. Wood leads a Web and networking technologies research group at Kodak. He received a PhD in computer science from Cornell University. Wood is a member of the IEEE Computer Society. Contact him at [mark.d.wood@kodak.com](mailto:mark.d.wood@kodak.com).*